

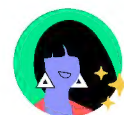


Implementando a transferência de dados com DTO

Durante o último módulo, nós construímos o backend de nossa API CRUD, disponibilizando uma série de endpoints que serão, em breve, consumidos pelo frontend de nosso projeto integrado. Se você testou cada um dos endpoints da API (caso ainda não tenha testado, recomendo que o faça o quanto antes) deve ter reparado que alguns métodos, depois de inseridos os dados, retornam com erro, não exibindo nenhuma informação. Além disso, vou além: se você deu uma olhada nos logs para tentar entender os erros, reparou, entre outras, algumas linhas contendo as informações abaixo:

- Java.lang.StackOverflowError: null
- HttpMessageNotWritableException: Could not write JSON: Infinite recursion

Uma rápida pesquisa no Google nos indica algumas soluções ou sugestões de solução para esses erros. Contudo, antes de aplicar uma dessas muitas sugestões, é importante entendermos um comportamento default do Spring referente ao relacionamento entre entidades. Veja o JSON abaixo, obtido via Swagger, no método save do TarefaController:






```
{
  "tarefaId": 0,
  "tarefaTitulo": "string",
  "tarefaDescricao": "string",
  "tarefaInicio": "2022-09-30T18:21:18.152Z",
  "tarefaFim": "2022-09-30T18:21:18.152Z",
  "statusTarefa": {
    "statusTarefaId": 0,
    "statusDescricao": "string",
    "tarefa": "string"
  },
  "projeto": {
    "projetoId": 0,
    "projetoNome": "string",
    "projetoDescricao": "string",
    "projetoInicio": "2022-09-30T18:21:18.152Z",
    "projetoFim": "2022-09-30T18:21:18.152Z",
    "projetoStatus": true,
    "tarefas": [
      "string"
    ],
    "gerentes": [
      {
        "recursoId": 0,
        "recursoNome": "string",
        "recursoFuncao": "string",
        "projetos": [
          "string"
        ],
        "tarefa": "string"
      }
    ]
  },
  "recurso": {
    "recursoId": 0,
    "recursoNome": "string",
    "recursoFuncao": "string",
    "projetos": [
      "string"
    ],
    "tarefa": "string"
  }
}
```


TarefaController – JSON do método save

Repare, na string, que como chave é trazido o item projeto. Dentro do mesmo, além das chaves referentes aos dados dessa entidade, é trazida também a chave tarefas, assim como a chave gerentes. Veja dentro dessa última chave que também há uma chave tarefa. Esse comportamento, de referenciar as entidades de forma circular,

é um comportamento padrão do Spring quando anotamos as relações entre as entidades de forma bi-direcional. Além disso, tal comportamento é aplicado não só aqui, no Swagger, na sugestão da String JSON para cadastro de uma nova entidade, mas também nos métodos de recuperação de dados. Como exemplo, vejamos a seleção de uma tarefa. Ao recuperar seus dados, serão recuperados os dados do projeto à qual ela está vinculada. Ao recuperar os dados do projeto, serão recuperados os dados dos gerentes a ele vinculado. Ao recuperar esses dados do gerente, serão recuperados os dados dos projetos aos quais ele está vinculado. Percebeu o comportamento circular? Isso gera a tal "recursividade infinita" (infinite recursion).

Para lidar com o problema de recursão infinita, há algumas diferentes abordagens, como modificar o "fetch type", nas anotações de relacionamento, nas entidades; incluir anotações adicionais nas entidades, seja para ignorar os relacionamentos, como a @JsonIgnore, seja para modificar/limitar a recursividade, como as anotações @JsonManagedReference e @JsonBackReference, a nível de relacionamentos, ou a @JsonIdentityInfo, a nível de Entidade. Cada uma dessas abordagens nos ajuda a resolver o problema de recursividade, apresentando diferentes comportamentos quando aplicadas. Inclusive, eu sugiro que você as aplique e veja o resultado. Entretanto, há outra forma de lidar com a recursividade, que pode tanto ser usada de forma isolada, sozinha, como combinada com as demais soluções. Tratam-se dos DTOs - ou Data Transfer Objects, também chamados de VO (Value Objects).

O DTO, de forma resumida, nada mais é que uma classe Java utilizada para a transferência de dados em uma aplicação. Lembre-se que, em nossa API, quem atualmente cumpre esse papel são as

Entidades. Se não percebeu isso, note em cada método que transferimos sempre uma instância, ou lista, de uma ou mais entidades, seja para salvarmos um dado no banco, seja para recuperar e devolver ao cliente que está consumindo a API. Até aqui estamos trafegando pelas nossas entidades. Ao utilizarmos o DTO, podemos modificar esse comportamento, passando a trafegar-lo no lugar, ou concomitantemente, da entidade, com uma importante diferença: enquanto a entidade é mais rígida, já que é o mapeamento da tabela, colunas e relacionamentos do banco de dados e, em linhas gerais, refletirá sempre, com exatidão, tal estrutura, o DTO é flexível, podendo ser exatamente igual à entidade, ou diferente, contendo mais ou menos atributos, contendo relacionamentos com outros DTOs ou não, e assim por diante.

Para facilitar o entendimento, veja os códigos abaixo. No primeiro temos a entidade Tarefa, como já a conhecemos. No segundo, temos o DTO Tarefa. Perceba a diferença entre ambos.



```
//imports
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "tarefa_id")
private Integer tarefaId;

@Column(name = "tarefa_titulo")
private String tarefaTitulo;

@Column(name = "tarefa_descricao")
private String tarefaDescricao;

@Column(name = "tarefa_inicio")
private Instant tarefaInicio;

@Column(name = "tarefa_fim")
private Instant tarefaFim;

@OneToOne
@JoinColumn(name = "status_tarefa_id",
            referencedColumnName = "status_tarefa_id")
private StatusTarefa statusTarefa;

@ManyToOne
@JoinColumn(name = "projeto_id", referencedColumnName = "projeto_id")
Projeto projeto;

@OneToOne
@JoinColumn(name = "recurso_id", referencedColumnName = "recurso_id")
Recurso recurso;

//get´s e set´s
```

Entidade Tarefa

```
//imports
private Integer tarefaId;
private String tarefaTitulo;
private String tarefaDescricao;
private Instant tarefaInicio;
private Instant tarefaFim;
private Integer statusTarefaId;
private Integer projetoId;
private Integer recursoId;
//get´s e set´s
```

DTO Tarefa

Tendo em vista que, no banco de dados a tabela Tarefa recebe, como chaves estrangeiras, as colunas status_tarefa_id, recurso_id e projeto_id, o que precisamos, em termos práticos, para persistir uma nova tarefa no banco são os números inteiros que representam esses registros. Perceba que o DTO Tarefa possui exatamente a representação desses tipos de dados. Ou seja, diferente da Entidade Tarefa, onde é feita uma relação entre entidades (Tarefa x StatusTarefa; Tarefa x Projeto; Tarefa x Recurso), o que, na prática, implica que o JSON de persistência de uma nova tarefa precisará representar essa estrutura, como visto acima, na string retirada do Swagger, ao usarmos um DTO temos a flexibilidade de simplificar os dados que serão transitados. Para efeitos comparativos, veja abaixo como fica o JSON para cadastro de uma nova Tarefa ao utilizarmos o seu DTO (conforme definido anteriormente):

```
{
        "tarefaId": 0,
        "tarefaTitulo": "string",
        "tarefaDescricao":"string",
        "tarefaInicio" : "2022-09-30T18:21:18.152Z",
        "tarefaFim": "2022-09-30T18:21:18.152Z",
        "statusTarefaId": 0,
        "projetoId": 0,
        "recursoId": 0
}
```

String JSON do DTO Tarefa

Conforme comentado, a utilização do DTO simplificou os dados transferidos para cadastro de uma nova tarefa. Esse é apenas um exemplo básico de tudo o que podemos fazer ao trafegarmos DTOs em lugar de Entidades em nossa API. Nesse sentido, cabe ressaltar

que também podemos manter a relação vista entre Tarefa e as demais entidades com as quais se relaciona, ao usarmos O. Entretanto, a diferença aqui é que devemos criar também os DTOs correspondentes às demais entidades, para que a relação seja sempre entre DTO x DTO e nunca entre DTO x Entidade. Para ficar mais claro, veja abaixo o exemplo, no fragmento de código, onde relacionamos o DTO de Tarefa com o DTO de Recurso:

```
//imports
private Integer tarefaId;
private String tarefaTitulo;
private String tarefaDescricao;
private Instant tarefaInicio;
private Instant tarefaFim;
private Integer statusTarefaId;
private Integer projetoId;
private Integer recursoId;
private RecursoDTO recursoDTO;
//get´s e set´s
```

Fragmento do DTO Tarefa

No fragmento acima foi mantido o atributo recursoId e acrescentado o recursoDTO, instância do DTO de Recurso, relacionando-o com o DTO de Tarefa. Normalmente, optaríamos por realizar esse relacionamento utilizando ou o integer recursoId ou a instância recursoDTO. Porém, o DTO é tão flexível que nos permite manter os dois e utilizá-los quando e como quisermos. Agora, veja o exemplo do que NÃO deve ser feito, ou seja, relacionar um DTO com uma Entidade:

```
//imports
private Integer tarefaId;
private String tarefaTitulo;
private String tarefaDescricao;
private Instant tarefaInicio;
private Instant tarefaFim;
private Integer statusTarefaId;
private Integer recursoId;
private RecursoDTO recursoDTO;
private Projeto projeto;

//get´s e set´s
```



Fragmento do DTO Tarefa

Nesse último exemplo foi incluída uma instância da entidade Projeto no DTO Tarefa. Embora funcione, sem retornar nenhum tipo de erro, devemos sempre evitar misturar entidades e DTOs. Logo, fica aqui o reforço: relacione entidades com entidades e DTOs com DTOs. Isso traz mais clareza ao código e evita confusões no momento de transitar os dados entre as camadas da API.

Antes de continuarmos, vamos rever e resumir o que tratamos até aqui sobre DTO:

- Servem para transitarmos dados na API – tanto para a recuperação quanto para persistência;
- Substituem as entidades, na transição de dados, fornecendo maior flexibilidade;
- São uma forma de resolver o problema de recursividade infinita, proveniente da relação bi-direcional entre entidades.

Sobre os pontos acima, precisaremos falar um pouco mais sobre o primeiro, ou seja, a transição de dados na recuperação ou persistência dos mesmos. Como já vimos, o Spring transita, desde o controller até o repositório, chegando por fim ao banco de dados, e vice-versa, instância ou instâncias de uma entidade e seus relacionamentos, quando existirem. Logo, não é necessário fazermos nenhuma configuração ou tratamento para que isso aconteça. Ao recuperarmos um dado, por exemplo, o repositório, através de seus métodos pré-definidos, como findAll, retorna uma lista de registros do banco representada por uma coleção de instâncias da entidade em questão. Da mesma forma, quando vamos persistir um novo dado, o fazemos através de uma string JSON cujos atributos representam a instância de uma entidade e seus relacionamentos. Se tudo isso acontece de forma simples quando usamos Entidades, o uso de DTO implica em alguns passos extras. Falaremos sobre isso a partir de agora.

A API que construímos até aqui foi toda configurada para a utilização de Entidades. Isso nos permite fazer uso de recursos como o ORM (Mapeamento Objeto Relacional) para a representação da estrutura do banco em classes Java, onde anotações Spring fazem com que isso seja possível, além de outras funcionalidades. Ao utilizarmos DTO, partiremos do princípio de que essa estrutura já criada não deverá ser modificada, visto que um dos objetivos de utilizarmos um framework, como o Spring, é justamente aproveitarmos as facilidades que o mesmo nos proporciona. Por outro lado, precisaremos, então, criar alguns códigos extras, permitindo assim que modifiquemos nossa API para trafegar DTOs, de maneira total ("substituindo" todas as Entidades por DTOs) ou parcial. Tal mudança consiste na "conversão" entre uma instância de DTO na instância da Entidade correspondente, e vice-versa. Veja o fluxo a seguir, onde o processo de cadastro de

uma nova tarefa é demonstrado já com a utilização do DTO no lugar da Entidade:



1. Controller recebe uma instância do DTO Tarefa (no método "save", por ex.);

2. Controller envia a instância de TarefaDTO para o TarefaService;

3. O serviço de Tarefa converte a instância de TarefaDTO numa instância da entidade Tarefa;

4. TarefaService envia a instância da entidade Tarefa para o TarefaRepository;

5. TarefaRepository persiste o dado no banco.

Da mesma forma, quando o endpoint de recuperação de uma tarefa (o "findById", por ex.) é chamado, teremos o seguinte processo:

1. Controller recebe a solicitação de recuperação de uma instância de Tarefa (findById);

2. Controller envia a solicitação para o TarefaService;

3. O serviço de Tarefa solicita ao Repositório de Tarefa a instância de Tarefa;

4. O repositório recupera o dado do banco e, **<u>no formato de Entidade</u>**, retorna a instância de Tarefa para o TarefaService;

5. TarefaService converte a instância da entidade Tarefa numa TarefaDTO;

6. TarefaService devolve o dado da Tarefa, como TarefaDTO, para o TarefaController;

7. TarefaController devolve o resultado para o cliente solicitante.

Veja nos fluxos que, conforme já mencionado, mantemos nossa API, em relação à camada que interage com o banco de dados, ou seja, a camada Repository, como já configurada anteriormente, ou seja, transitando Entidades (a fim de fazermos uso dos benefícios que o Spring nos fornece). A partir da camada seguinte, a camada Service (que guarda nossas regras e inteligência de negócio), temos um processo extra onde a instância da entidade precisa ser convertida na instância do DTO, tanto para os dados recebidos para serem persistidos, quanto para os recuperados e que serão devolvidos na sequência. Na prática, embora represente linhas de código adicionais, essa mudança na API para uso do DTO compensa esse trabalho extra pelos benefícios e facilidades que o mesmo representa. Outro ponto aqui é que não existe uma obrigatoriedade de modificarmos toda a API para usar, em todas as situações, o DTO no lugar da Entidade. Isso deve ser analisado caso-a-caso, pois em casos simples, de entidades sem relacionamentos, ou com relacionamentos de pouca recursividade, devemos manter o uso da Entidade em si, já que a conversão para DTO não representaria grandes vantagens.

Após falar, conceitualmente, da mudança no fluxo de nossa API, veja abaixo, na prática, no código-fonte, o serviço de Tarefa, onde o método “saveTarefa” foi modificado para receber uma TarefaDTO, chamar a função de conversão para Entidade, e, por fim, enviá-la para o repositório.



```
//...

public Tarefa saveTarefa(TarefaDTO tarefaDTO) {
        //Cria uma nova instancia da entidade Tarefa,
        //que recebe o resultado da conversao do DTO
        Tarefa novaTarefa = converteDTOParaEntidade(tarefaDTO);
        //Salva a nova Tarefa e a retorna
        return tarefaRepository.save(novaTarefa);
}

private Tarefa converteDTOParaEntidade(TarefaDTO tarefaDto) {
        Tarefa tarefa = new Tarefa();
        //tarefa.setProjeto(tarefaDto.getProjetoId());
        //tarefa.setRecurso(tarefaDto.getRecursoDTO());
        //tarefa.setStatusTarefa(tarefaDto.getStatusTarefaId());
        tarefa.setTarefaDescricao(tarefaDto.getTarefaDescricao());
        tarefa.setTarefaFim(tarefaDto.getTarefaFim());
        tarefa.setTarefaInicio(tarefaDto.getTarefaInicio());
        tarefa.setTarefaTitulo(tarefaDto.getTarefaTitulo());

        return tarefa;
}

//...
```

Fragmento do TarefaService

Perceba que o código ainda está incompleto – veja o método converteDTOParaEntidade, em que há 3 linhas de código comentadas. Em instantes veremos o porquê. Antes, veja que a estrutura existente anteriormente, no método saveTarefa, foi modificada. Primeiro, porque o método agora recebe uma TarefaDTO e não mais uma Tarefa (entidade). A seguir, e considerando que o Repository só trabalha com entidade, é realizada a conversão do DTO numa Entidade. Por fim, essa nova entidade é devolvida ao final do método. Outra mudança aqui é a criação do método de conversão. Nele foi criada uma instância da entidade Tarefa e, através dos métodos “set”, foram atribuídos valores para seus atributos, a partir da instância de TafefaDTO recebida. Para atributos simples, essa conversão também é

simples: setamos no atributo da entidade o atributo correspondente do DTO. Entretanto, quando falamos de atributos que são instâncias de outras entidades, isso se torna um pouco mais complicado. Lembre-se que um DTO se relaciona com outro DTO – da mesma forma que uma Entidade com outra Entidade. Logo, é por isso que há 3 linhas comentadas no fragmento de código. Vejamos a primeira:

```
//tarefa.setProjeto(tarefaDto.getProjetoId());
```

Explicando em detalhes: a entidade Tarefa se relaciona com a entidade Projeto. Consequentemente, há um médoto "setProjeto", que recebe a instância da entidade Projeto. Em nosso DTO, mapeamos essa relação apenas como um número inteiro, representando o código(id) do projeto. Nesse caso, há uma incompatibilidade de tipo de dado, já que não podemos "setar" na entidade tarefa um Projeto a partir de um inteiro. Mantendo esse mapeamento em TarefaDTO, e para conseguir setar uma instância do projeto, precisaremos de um passo adicional, que é recuperar uma instância de Projeto, a partir de seu Id para, então, setá-la em Tarefa. Para facilitar, veja como ficará o nosso código nesse caso:

```
    //...

    private Tarefa converteDTOParaEntidade(TarefaDTO tarefaDto) {
        Tarefa tarefa = new Tarefa();

        //Recuperando uma instancia de Projeto,
            //  via repository, a partir de seu id
        Projeto projeto =
            projetoRepository.findById(tarefaDto.getProjetoId())
                        .orElse(null);

        //Setando em tarefa a instancia recuperada de projeto
        //  (ou NULL, caso nao existe projeto com o id fornecido)
        tarefa.setProjeto(projeto);

        //tarefa.setRecurso(tarefaDto.getRecursoDTO());
        //tarefa.setStatusTarefa(tarefaDto.getStatusTarefaId());
        tarefa.setTarefaDescricao(tarefaDto.getTarefaDescricao());
        tarefa.setTarefaFim(tarefaDto.getTarefaFim());
        tarefa.setTarefaInicio(tarefaDto.getTarefaInicio());
        tarefa.setTarefaTitulo(tarefaDto.getTarefaTitulo());

        return tarefa;
    }

    //...
```

Fragmento do TarefaService::converteDTOParaEntidade

O código está comentado, informando o que está sendo feito: a partir do id do projeto, constante em tarefaDto, e usando o método "findById" do repositório de projeto (que precisa ser adicionado, anotado com @Autowired, aqui no TarefaService) é recuperada, caso exista no banco de dados, a instância correspondente da entidade Projeto. A seguir, tal instância é, então, setada em Tarefa.

Olhando ainda para o método de conversão, o mesmo processo acima deverá ser realizado para os métodos "tarefa.setRecurso" e "tarefa.setStatusTarefa". Você poderá fazer esse trabalho como atividade prática desse módulo.

Para essas tarefas de conversão (Entidade para DTO e vice-versa) você poderá utilizar os códigos apresentados aqui – a princípio mais simples, a fim de destacar os princípios envolvidos no processo em questão, ou outros mais elaborados, como Stream´s Java ou classes e bibliotecas específicas de conversão.

Por fim, na conclusão deste módulo e em complemento aos códigos vistos até aqui, veremos também como converter uma Entidade em um DTO. Em linhas gerais, trata-se do mesmo processo, que faz uso do mesmo fluxo já visto anteriormente: cada atributo da entidade deve ser convertido no atributo correspondente do DTO. Ainda na classe TarefaService, veja como ficaria o método “converteEntidadeParaDTO”:

```
//...

private TarefaDTO converteEntidadeParaDTO(Tarefa tarefa) {
	TarefaDTO tarefaDto = new TarefaDTO();

	//Para evitar uma excecao nullpointer,
		//  precisamos verificar se foi
	//  recebida uma instancia da entidade Projeto a partir
		//  da entidade Tarefa.
	//  Caso positivo, pegamos o seu id a partir de
		//  tarefa.getProjeto.
	//  Caso negativo, setamos o projetoId como null
	if(null != tarefa.getProjeto())

				tarefaDto.setProjetoId(
					tarefa.getProjeto().getProjetoI
		d()
				);
	else
		tarefaDto.setProjetoId(null);

	//Para setar o recursoDTO, precisariamos recuperar seus dados
	//  a partir do tarefa.getRecursoId, recuperando
		//  primeiro a instancia da
	//  entidade e, a seguir, convertendo tambem para DTO.
	//Entretanto, aqui vamos usar o id do recurso, setado abaixo
	//tarefaDto.setRecursoDTO(RecursoDTO);

	tarefaDto.setRecursoId(tarefa.getRecurso().getRecursoId());
	tarefaDto.setStatusTarefaId(
				tarefa.getStatusTarefa().getStatusTarefaId()
				);

	tarefaDto.setTarefaDescricao(tarefa.getTarefaDescricao());
	tarefaDto.setTarefaFim(tarefa.getTarefaFim());
	tarefaDto.setTarefaInicio(tarefa.getTarefaInicio());
	tarefaDto.setTarefaTitulo(tarefa.getTarefaTitulo());

	return tarefaDto;
}

//...
```

Fragmento do TarefaService::converteEntidadeParaDTO

Outro método que recebe dados é o de update. Logo, é interessante modificá-lo para que também transite DTO no lugar da Entidade. Para isso, pode ser utilizada a mesma estratégia vista com o método save, em relação à conversão entre entidade x DTO,

ou fazer uma implementação de conversão diretamente no método. Veja um exemplo abaixo:



```
public TarefaDTO updateTarefa(Integer id, TarefaDTO tarefaDto) {
        Tarefa tarefaAtualizado =
                        tarefaRepository.findById(id).orElse(null);
        if(tarefaAtualizado != null) {
                //Recuperando uma instancia de Projeto,
                        // via repository, a partir de seu id
                Projeto projeto =
                                        projetoRepository.findById(
                                                tarefaDto.getProjetoId()).orElse(null);

                //Setando em tarefa a instancia recuperada de projeto
                //  (ou NULL, caso nao existe projeto com o id fornecido)
                tarefaAtualizado.setProjeto(projeto);

                if(null != tarefaDto.getRecursoId()) {
                        Recurso recurso = recursoRepository.findById(
                                                tarefaDto.getRecursoId()).orElse(null);

                        //Caso exista recurso com o id recebido, seta seu valor.
                        //  Caso nao, seta como null
                        if(null != recurso)
                                tarefaAtualizado.setRecurso(recurso);
                        else
                                tarefaAtualizado.setRecurso(null);
                }else if(null != tarefaDto.getRecursoDTO()) {
                        //Para evitar um nullpointer exception,
                                // verifica se o RecursoDTO foi recebido

                        //Recupera, via repository, a instancia
                                // de recurso, passando pelo RecursoDTO e chegando
                        //  ao atributo RecursoId
                        Recurso recurso = recursoRepository.findById(
                                                        tarefaDto.getRecursoDTO()
                                                        .getRecursoId()).orElse(null);
                        //Caso exista recurso com o id recebido, seta seu valor.
                        //  Caso nao, seta como null
                        if(null != recurso)
                                tarefaAtualizado.setRecurso(recurso);
                        else
                                tarefaAtualizado.setRecurso(null);
                        }else {
                                tarefaAtualizado.setRecurso(null);
                        }

                        //Caso exista StatusTarefa com o id recebido,
                                // seta seu valor. Caso nao, seta como null
                        StatusTarefa statusTarefa =
                                                statusTarefaRepository.findById(
                                                tarefaDto.getStatusTarefaId()).orElse(null);
                        if(null != statusTarefa)
                                tarefaAtualizado.setStatusTarefa(statusTarefa);
                        else
                                tarefaAtualizado.setStatusTarefa(null);

                                tarefaAtualizado.setTarefaDescricao(tarefaDto.getTarefaDescricao());
        tarefaAtualizado.setTarefaFim(Instant.ofEpochMilli(tarefaDto.getTarefaFim()));
        tarefaAtualizado.setTarefaInicio(Instant.ofEpochMilli(
                        tarefaDto.getTarefaInicio()));
                        tarefaAtualizado.setTarefaTitulo(tarefaDto.getTarefaTitulo());
                        return
                                                converteEntidadeParaDTO(
                                                        tarefaRepository.save(tarefaAtual
                                izado));
        }else {
                return null;
                }
}
```

Conforme já mencionado, utilizar DTO simplifica a troca de dados, uma vez que é possível simplificar e diminuir a quantidade dos mesmos. Nesse sentido, veja o DTO a seguir:

```
public class RecursoIdNomeDTO {
	private Integer recursoId;
	private String recursoNome;

	public Integer getRecursoId() {
		return recursoId;
	}

	public void setRecursoId(Integer recursoId) {
		this.recursoId = recursoId;
	}

	public String getRecursoNome() {
		return recursoNome;
	}

	public void setRecursoNome(String recursoNome) {
		this.recursoNome = recursoNome;
	}
}
```

Aqui temos um DTO simples que retorna apenas o id e o nome do Recurso. Tal DTO é bastante útil para o frontend, para popular componentes "select", onde apenas esses dados são necessários. Assim como para recurso, você pode implementar outros DTOs como esses, ou então, DTOs para a população de listas de dados. Aqui tem um exemplo desse, de um DTO de listagem de Tarefas:

```
public class ListagemTarefaDTO {
	private Integer tarefaId;
	private String tarefaTitulo;
	private String tarefaDescricao;
	private Instant tarefaInicio;
	private Instant tarefaFim;
	private StatusTarefaDTO statusTarefaDTO;
	private ProjetoDTO projetoDTO;
	private RecursoDTO recursoDTO;

	//Gets e Sets
}
```

Os códigos vistos ao longo desse módulo encontram-se no repositório abaixo, onde o projeto que desenvolvemos no módulo 14 foi atualizado para utilização de DTO. Você deverá utilizar esse projeto como backend final do projeto integrado, conectando o frontend, que desenvolveremos no próximo módulo, a ele.

https://github.com/FaculdadeDescomplica/pratica_integradora_tecnologias_disruptivas/tree/main/modulo15/backend **(Acesso em 31/10/2022)**

Com esses exemplos práticos de utilização de DTO, para necessidades típicas na integração com o frontend, chegamos ao final do módulo 15, ao longo do qual foi discutida a otimização da API, assim como uma das estratégias para contornar a recursividade infinita de entidades, a partir da utilização do DTO.

Atividade Extra

Além de entender como usar o DTO, é importante também conhecer um pouco mais sobre os conceitos envolvidos nos objetos de transferência de dados. Nesse link abaixo você pode conhecer um pouco mais esse padrão:

https://www.baeldung.com/java-dto-pattern **(Acesso em 31/10/2022)**

Referência Bibliográfica



WALLS, Craig. **Spring Boot in Action**. Shelter Island, NY: Manning Manning, 2016.

Atividade Prática Módulo 15

Título da Prática: Conversão entre DTO e Entidade.

Objetivos: Converter a instância de uma classe DTO na instância de uma classe de Entidade.

Materiais, Métodos e Ferramentas: IDE Spring Tools Suite, Docker.

Atividade Prática

Ao longo do módulo 15 vimos o processo de conversão entre DTO e Entidade, e vice-versa. Para praticar tal processo, você deverá complementar o método "converteDTOParaEntidade", da classe TarefaService, implementando os passos faltantes, no fragmento de código a seguir, referentes a "tarefa.setRecurso" e "tarefa.SetStatusTarefa".



```
//...

private Tarefa converteDTOParaEntidade(TarefaDTO tarefaDto) {
       Tarefa tarefa = new Tarefa();

       //Recuperando uma instancia de Projeto,
//  via repository, a partir de seu id
       Projeto projeto =
projetoRepository.findById(tarefaDto.getProjetoId())
.orElse(null);

       //Setando em tarefa a instancia recuperada de projeto
       //  (ou NULL, caso nao existe projeto com o id fornecido)
       tarefa.setProjeto(projeto);

       //tarefa.setRecurso(tarefaDto.getRecursoDTO());
       //tarefa.setStatusTarefa(tarefaDto.getStatusTarefaId());
       tarefa.setTarefaDescricao(tarefaDto.getTarefaDescricao());
       tarefa.setTarefaFim(tarefaDto.getTarefaFim());
       tarefa.setTarefaInicio(tarefaDto.getTarefaInicio());
       tarefa.setTarefaTitulo(tarefaDto.getTarefaTitulo());

       return tarefa;
}

//...
```

Fragmento do TarefaService::converteDTOParaEntidade

OBS: você deverá usar a mesma estratégia vista ao longo do módulo 15, para setar o Projeto, no “tarefa.setProjeto”.

Gabarito Atividade Prática

A resolução dessa atividade consistirá no complemento do fragmento de código acima, onde mais de uma solução poderá ser aplicada, sendo uma delas a apresentada abaixo, que pode servir como gabarito para fins de comparação pelo aluno.

```
@Autowired
RecursoRepository recursoRepository;

@Autowired
StatusTarefaRepository statusTarefaRepository;

//...

private Tarefa converteDTOParaEntidade(TarefaDTO tarefaDto) {
Tarefa tarefa = new Tarefa();

//Recuperando uma instancia de Projeto via repository, a partir de seu id
Projeto projeto =
projetoRepository.findById(
tarefaDto.getProjetoId()).orElse(null);

//Setando em tarefa a instancia recuperada de projeto
//  (ou NULL, caso nao existe projeto com o id fornecido)
tarefa.setProjeto(projeto);

//Para evitar um nullpointer exception,
//  verifica se o RecursoDTO foi recebido
if(null != tarefaDto.getRecursoDTO()) {
        //Recupera, via repository, a instancia de recurso,
//  passando pelo RecursoDTO e chegando  ao atributo RecursoId
        Recurso recurso =
recursoRepository.findById(tarefaDto.getRecursoDTO()
.getRecursoId()).orElse(null);

        //Caso exista recurso com o id recebido, seta seu valor.
        //  Caso nao, seta como null
        if(null != recurso)
                tarefa.setRecurso(recurso);
        else
                tarefa.setRecurso(null);
}else {
        tarefa.setRecurso(null);
}

//Caso exista StatusTarefa com o id recebido, seta seu valor.
//  Caso nao, seta como null
StatusTarefa statusTarefa =
statusTarefaRepository.findById(
tarefaDto.getStatusTarefaId()).orElse(null);

if(null != statusTarefa)
        tarefa.setStatusTarefa(statusTarefa);
else
        tarefa.setStatusTarefa(null);

tarefa.setTarefaDescricao(tarefaDto.getTarefaDescricao());
tarefa.setTarefaFim(tarefaDto.getTarefaFim());
tarefa.setTarefaInicio(tarefaDto.getTarefaInicio());
tarefa.setTarefaTitulo(tarefaDto.getTarefaTitulo());

return tarefa;
}
//...
```

Fragmento do TarefaService::converteDTOParaEntidade

Ir para exercício